University of Portsmouth

School of Computing

Mobile App for Learning Basic Algorithm Construction

Project Report

Authors

Jack Turner

Supervisor

Matthew Poole

July 13, 2021

Abstract

Learning to program has long been a difficult aspect of education in computer science. With the advent of more interactive methods of teaching in a modern era, there now exist many possible solutions to this issue. This project puts forward a new interactive puzzle game that takes inspiration from Blocks Based Programming and Parsons Problems. These two existing methods help educate and assess novices in the field of programming. This game's aim is to focus a learner on the paradigms of an algorithm and guide them through completing puzzles to assist in reinforcing their knowledge. This report delves into the background of the project and the processes undertaken in its completion to produce the final version of the puzzle game.

Contents

1 Introduction 1

1.1 Background 1

1.2 Aims 2

1.3 Structure of Document 3

2 Literature Review 5

2.1 Interactive Technology in Education 5

2.2 Block Based Programming 7

2.3 Parsons Problems 8

2.4 Summary of Findings 10

3 Methodology and Management 11

3.1 Life-cycle 11

3.2 Timescale 12

3.3 Risks 12

4 Requirements 15

4.1 Requirements Gathering 15

4.2 Functional Requirements 15

4.3 Non-Functional Requirements 20

5 Design 22

5.1 Navigation 22

5.2 Layout Design 23

5.3 Appearance 30

5.4 Drag and Drop 32

5.5 Problem Design 33

6 Implementation 35

6.1 Requirements Changes 35

6.2 Implementation Difficulties 37

6.3 Testing Methodology 40

6.4 Testing Table 41

7 Evaluation 43

7.1 Evaluation against Functional Requirements 43

7.2 Summary of Evaluation against Requirements 48

7.3 Evaluation against Non-Functional Requirements 49

8 Conclusion 51

1 Introduction

The basis of this project is to produce a mobile app designed to guide novices in programming on basic algorithm construction. This report describes the steps taken to produce the final app prototype including its design and implementation. This report also details the reasoning behind many of the design decisions made during development and evidences the need for this project through a literature review. Section 1.1 of the chapter will cover the background of the project and the motivations on why this project was undertaken. Section 1.2 will give a brief description on the aims of the project and Section 1.3 will give an overview on what content is covered within each chapter of this report.

1.1 Background

Programming, at its heart, is a taught skill. It requires the use of a persons' skill at problem solving and logical thinking to create algorithms and programs to accomplish tasks. It is for these reasons many consider it a difficult subject and why it is taught in many schools, universities and online courses. Despite its difficulty, Computer Science maintains high numbers of enrolments with 88,025 students enrolled onto Computer Science courses in UK universities in 2017/2018 (Higher Education Statistics Agency, 2019). With so many students and such hard subject matter, it is only natural that various methods of teaching are developed to help educate students on how to attain and improve their skills at programming. Two examples of this methodology include Block Based Programming (BBP) and Parsons Problems.

BBP systems have the user linking snippets of code to each other by dragging blocks into a work-space. Scratch (https://scratch.mit.edu/) is an example of Block Based Programming that can be used to create animations or simple games. BBP is a fantastic way of teaching novices the basics of creating their own programs without the worry of learning syntax, data structures or complex algorithms. However, this method lacks the fidelity to allow the creation of more intricate programs which thoroughly challenges a users understanding of programming.

Parson's Problems (PP) are an informal teaching method which involves the user organising randomly shuffled lines of code to produce the correct answer. Distractors, lines that should not be included in the final solution, can be used to increase the difficulty of a problem. PP are a great way of introducing short puzzles for a user to solve that give instant feedback. Similar to BBP, PP don't require any prior knowledge of a particular languages' syntax. A limitation to this type of problem is that the user isn't given specific feedback if their given answer is incorrect outside of being told that the lines are in the wrong order. This type of feedback isn't helpful to the user in the long term as it doesn't guide them to understand where they went wrong.

Both of these methods of teaching lack the capacity to provide challenging, quick to solve puzzles, that test a users comprehension and ability in problem solving. It was for this reason this project was devised, putting forward a method of informal teaching that helps develop both of these skills. PP initially inspired this project, due to its simple but versatile concept. This was then further developed, taking cues from BBP to include the use of blocks of code. The final design then, involves the user moving blocks of code into blank spaces. These blank spaces represent missing code from an algorithm. The user's task is to move the correct blocks into their correct spaces to form a complete algorithm that returns a desired result.

1.2 Aims

The primary objective of this project is to produce an app prototype that serves as an informal teaching aid for novice programmers in the construction and comprehension of basic to advanced algorithms. This project has a timescale of nine months, currently planned for September 2019 to May 2020 at this point. At the end of this timescale, all artefacts including but not limited to the final report and app prototype must be complete.

This project is inspired by other methods of teaching, specifically BBP and PP. The app should be an informal teaching environment that is fun and engaging to the user. This will take the form of a problem-solving game where the user is presented an incomplete algorithm with missing portions of code. With the provided environment, the user is then able to drag blocks of code into these blank spaces to complete the algorithm. The user solves the problem by putting blocks of code into their correct spaces. The app should focus on the user's comprehension of the problem and not the algorithms' syntax as such. However, the algorithm and blocks of code should follow a particular languages' syntax in order to liken it more to a real life scenario. If a user gets a question incorrect, they should be given feedback specific to where they went wrong. The user should not get feedback simply telling them they have the blocks in the wrong place unless they have a simple logic error. An example of a logic error being that they have used an incorrect constant that would lead to a different result than expected at runtime.

There were many risks to the accomplishment of this project. Significant examples of risks include the limited duration of the project, lack of resources and impact from data loss. The hard deadline of the project, with small chance of an extension, means that there was a risk that the project will not be completed if time working on the project is ill-managed. Since I was the only person working on the project, I had to make sure there are measures in place to mitigate risk to the project if I am unwell or otherwise unable to work on the project. All work towards the project must be backed up to protect against any risk of data loss. Precautionary measures include using cloud storage and keeping local copies on multiple computers.

1.3 Structure of Document

This document will consist of a total of seven chapters further to the introduction, that each describe the processes undertaken in the development of the project. These chapters are as follows:

In Chapter 2, a literature review is conducted, analysing interactive mediums used in education and their benefits, Block Based Programming and Parsons Problems.

Chapter 3 will describe this project's methodology and how it was managed. This will include a further look into the projects' implementation life-cycle, risks and timescale.

Chapter 4 contains all the functional and non-functional requirements for the app, giving a representation of what is expected from the final result.

Chapter 5 covers the navigation, layout designs and theory behind the problem designs.

In Chapter 6, the implementation phase of the project is explored. This includes summarising any difficulties found during development and incorporating changes that had to be made to the requirements to suit new ideas and feedback. This chapter also details the strategies used to test the final product.

Chapter 7 relates the final product to the requirements and evaluates whether it has met these criteria.

Chapter 8 will finalise the report in an overall conclusion with reflections on the success of the project and suggestions for the future of this project.

2 Literature Review

This chapter covers background research conducted to reinforce the need for this project. Initially, a summary on the need of interactive technology to teach students is given in Section 2.1, with an example of a generic platform that is used to reinforce learning in all conceivable topics. Section 2.2 explores the use of Block Based Programming to teach programming paradigms and concepts with a closer look into Scratch and the effects of its use. Section 2.3 delves into the advantages and disadvantages of Parsons Problems as well as it's history and possible future. Finally, Section 2.4 will give a brief summary of the findings and will relate it to the inspirations behind this project.

2.1 Interactive Technology in Education

Learning to program is a difficult task, traditional methods of education such as lectures are not enough in the modern era and more interactive environments such as games or online quizzes are becoming more prevalent.

There are many aspects of programming that make it difficult for novices. bt Rahim, Zaman, Ahmad, and Ali (2018) concludes in their research that the main areas of difficulty that students face when learning programming are their lack of problem-solving skills and understanding of basic programming concepts. To solve this, bt Rahim et al. (2018) recommends that teachers make use of system simulation and games to reinforce existing teaching methods. Recommendations on using visual stimuli to aid in teaching programming concepts is not a new idea, a study conducted 16 years prior to bt Rahim et al. (2018) also suggests using a visualisation tool to aid in teaching concepts that students found to be most difficult (Milne and Rowe, 2002).

With modern technological advances, there now exist many interactive tools to aid in teaching a wide array of subjects, not just programming. Battistella and von Wangenheim (2016) reviewed 107 games designed to teach topics related to computer science such as software engineering and programming. They found that these reviewed games have a "promising approach to increase learning effectiveness" and were found to be mostly used to reinforce and review prior knowledge. Studies by Rahman and Paudel (2018) for example, showed students using interactive methods through using an eBook and paired programming displayed higher engagement with the subject and such methods suggest “more-effective learning”.

Battistella and von Wangenheim (2016) warns that games used to educate students need to follow instructional and game theory in order to be effective. This is confounded by Kalelioglu and Gülbahar (2014) that, despite conducting a study involving young children compared to University students, also concluded by suggesting that providing a learning environment isn’t enough and that guidance and support by teachers is also required. Though this same study points out the merits of using such a medium in that it built on a students self-confidence in their own problem solving skills. What all these studies have in common is that they see the potential in the use of interactive mediums to educate students. However, some of these studies also suggest that a more guided approach alongside these mediums is also needed to reinforce and guide students and these mediums alone are not enough.

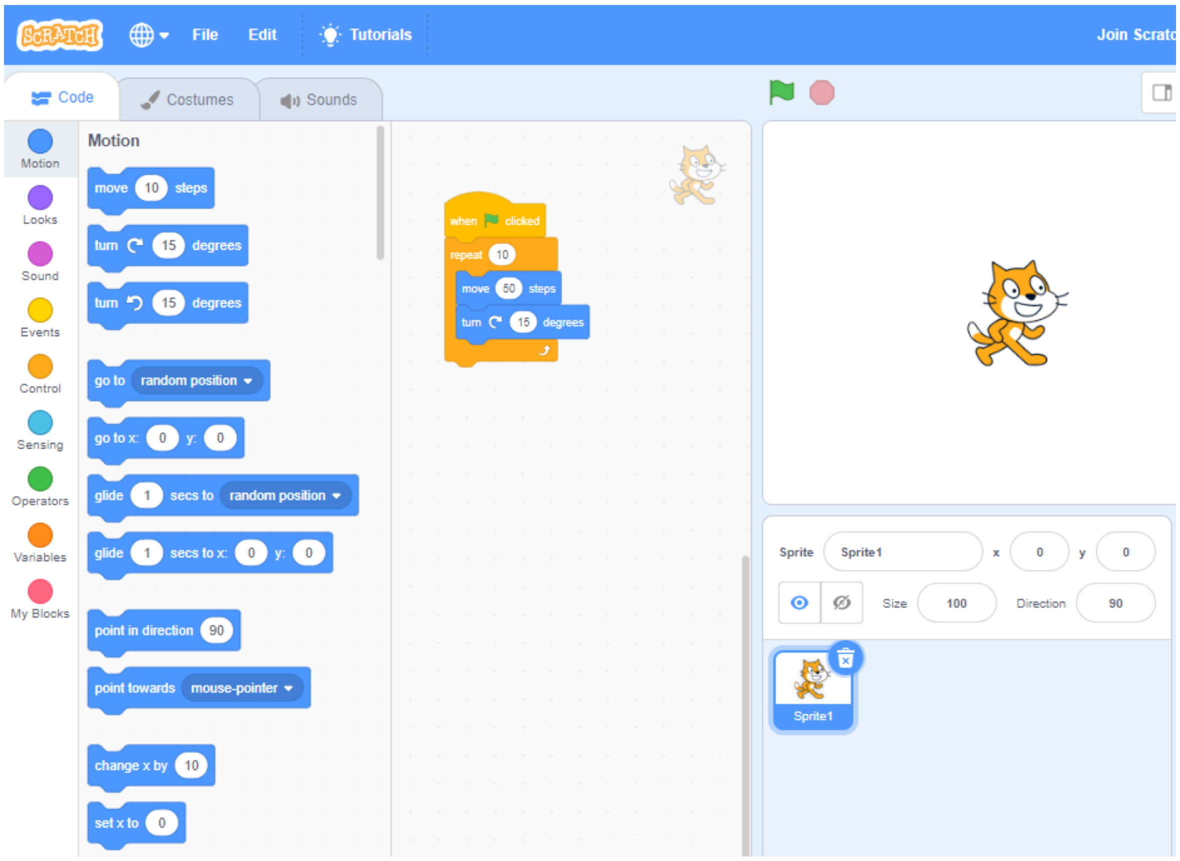
As mentioned, there exist many interactive mediums to help in education. An example of a generic platform is “Kahoot!”, where users can access user-generated multiple choice quizzes via the Kahoot app or web browser (kahoot.com). This tool can be used to assess and reinforce knowledge on any subject, since its quizzes are created by it’s users. Gibson (2015) mentions that they use it in their classroom to “spark interest at the beginning of a topic/unit” and assess the level of their students before starting a new topic. Kahoot is immensely popular with 1 billion cumulative players as of 2017 (Keane, 2017). Due to its generic nature, Kahoot is not tailored to any specific topic or subject area and may not lend itself well to testing students on a particular skill. In the case of teaching programming, Kahoot is unable to get students to write or experiment with code. It could be used to test users on their ability to trace code however, though this type of test is examined and found to be lacking as discussed in Section 2.3.

2.2 Block Based Programming

Mentioned in Section 2.1, programming concepts are considered by students to be one of the harder paradigms that they find most difficult (bt Rahim et al., 2018). A guide to teaching programming mentions that “breaking the problem down into smaller single-concept pieces can reduce the cognitive load to something manageable” (N. C. Brown and Wilson, 2018). Both of these points relate directly to Block Based Programming (BBP). One of the most popular examples of BBP is Scratch. This platform allows the user to create programs such as games or interactive stories through assembling command blocks to create code (Maloney, Resnick, Rusk, Silverman, and Eastmond, 2010). Scratch is intended to educate users in the basic concepts of programming through an easy-to-use interface. How Scratch relates to the two above points is that by using blocks, it can break down a problem into more manageable pieces and disregards the need for user to worry about concepts such as syntax. Instead it focuses more on the semantics of the code itself.

BBP is not without issue however, Repenning (2017) warns that removing syntax will only support a student so far and that a more “daunting challenge” is to support the students in learning programming pragmatics. This correlates Kalelioglu and Gülbahar (2014) who studied the impacts of using Scratch in a primary school classroom. Kalelioglu and Gülbahar (2014) concludes that Scratch had little effect on an average pupil’s ability to problem solve and that pupils required specific guidance from the teacher alongside using Scratch in order to improve sufficiently. This study only delved into the short term effects having only conducted the study over 5 weeks however, and is refuted by other studies including a study by Q. Brown et al. (2013). Q. Brown et al. (2013) conducted a similar study with pupils of the same age over a short length of time. When comparing a set of test groups against a control, they discovered a difference in the two groups abilities when tested. The groups that made use of Scratch in lessons showed a clear increase in marks compared to that of the control. This is supported by another study by Wang, Huang, and Hwang (2014) that showed a positive increase in attained marks by users that made use of Scratch.

Figure 1 is a screenshot of the workspace provided by Scratch. As shown, the user drags blocks from the left hand side into the workspace in the centre. In the given exam-



```
when [flag] clicked
repeat 10
    move 50 steps
    turn 15 degrees
```

Figure 1: Scratch

ple, the sprite will repeat 10 times the action of moving 50 paces and rotating 15 degrees clockwise. Scratch can be accessed at at https://scratch.mit.edu/projects/editor/.

2.3 Parsons Problems

Parsons Problems, initially developed by Dale Parson and Patricia Haden, are a form of puzzle that involves a learner rearranging mixed-up lines of code to form a complete algorithm (Parsons and Haden, 2006). These problems were designed to engage learners, provide immediate feedback and model good code to novices, amongst other reasons. Since the learners are being given the code, their focus is less on the syntax or semantics but on how the algorithms should run. This is similar to the mediums mentioned prior in that they are designed to be a more interactive alternative to traditional methods of assessment. In this case, Parsons Problems were designed as an alternate to “drill exercises” that Parsons and Haden (2006) considered to be boring and tedious. In their proposal of Parsons Problems, Ericson, Margulieux, and Rick (2017) gathered feedback on this approach and found that students believe these exercises were “useful and even fun”.

There have been many outside studies conducted that evaluate Parsons Problems. A study conducted by Ericson et al. (2017), suggested that using Parsons Problems to practise programming had the potential to be a more efficient method of learning compared to writing and fixing code while still providing the same improvement on a students skills. Denny, Luxton-Reilly, and Simon (2008) concluded that Parsons Problems were an excellent alternate to writing code style questions that still demanded the same set of skills. Denny et al. (2008) states that Parsons Problems are easier and less subjective to mark though does not go into detail about what feedback was given to the user if they answered a problem incorrectly. This issue was also discovered by Parsons and Haden (2006) in their early testings. Student feedback included wanting more detailed feedback about why their answer was incorrect. Parsons and Haden (2006) did not explore a solution to this problem in great depth outside of suggesting that “comments for each of the potential errors” be made for every problems. They later state that they doubt they would be able to “reliably identify a student’s conceptual misunderstanding”.

Denny et al. (2008) also delved into what makes a good Parsons Problem. Following student feedback, they discovered that more structure in the form of indentation, braces and indication into the number of lines made the problems far easier to understand. By applying more structure in Parsons Problems, they were able to identify “their obstacle or mistake by seeing structure”. The study also discovered that students found it easier to contextualise their solution when re-writing the lines of code in order. Denny et al. (2008) conducted the experiment with students answering the problems in traditional exam-style conditions, no electronic interface was used. Parsons and Haden (2006) did not discover this issue, potentially because they had used a web-based application with a basic user-interface. Instead, they received feedback that their user interface would be improved with colour, animation and sound. These types of improvements would effect the usability but would have little impact on the problems themselves.

2.4 Summary of Findings

In summary of what has been researched, using interactive methods of teaching to reinforce learning and practises is evidenced to be a potential benefit to users. This project then, puts forward a new interactive game that combines the structure of Parsons Problems and the interactivity of Block Based Programming. This game aims to allow users to practise their programming skills in testing their ability to trace algorithms and complete them to produce the correct output. From what has been discussed in the prior sections, the research proves that the result of this project, if it is successful, should be nothing but a positive to learners.

3 Methodology and Management

This chapter discusses how the project was managed and what methodologies were used to produce the final result. Section 3.1 discusses the life-cycle methodology of the project and why it was used. Section 3.2 provides a brief description of the timescale of the project, and the risk this poses to the final product. Lastly, Section 3.3 covers significant risks to the project and how they were managed to improve the projects' probability of success.

3.1 Life-cycle

Through the duration of the project, biweekly meetings were conducted between the project's supervisor and myself. During these meetings, progress and planning of the project was discussed as well as any feedback the supervisor had for the prototype app. Early on, it was decided that the project should take on an iterative life-cycle and would involve development cycles with meetings between them. Every two weeks, a cycle of adding new features to meet requirements and implementing changes based off of feedback would begin. At the end of a cycle, we would review the previous weeks efforts, the next two weeks would be planned and the next cycle would begin. This is similar to Sprints within Agile but in concept only since they share short development cycles with meetings in between but are better suited to large teams building already functional software.

To aid development, the requirements for the final prototype are separated into three categories: *Must Do's*, *Should Do's* and *Could Do's*. The specific requirements under these categories are detailed in Chapter 4. The implementation period of the project was organised such that development would initially focus on the first category, then the second and so forth. This methodology was used to promote the app having its core components completed before development of less critical components began.

Following the iterative process mentioned prior, these categories of requirements had to be considered when pre-planning each new development cycle. This adjusted the methodology to follow a model similar to Waterfall where each development phase had to be completed before moving onto the next, though it lacked Waterfall's hard deadlines. Soft deadlines, discussed in every cycle meeting, were instead implemented. This helped to increase the project's reactivity to incoming feedback where adding less important features was delayed in favour of improving/fixing existing critical functions.

3.2 Timescale

This project began in September 2019 and finished in May 2020, giving a total duration of nine months. During this time, this report, the final app prototype and a Project Initiation Document (PID) had to be completed. This short time frame to complete these three artefacts was a significant risk to the project, so time management was key. This was a key risk is due to the hard deadline of May 2020, no extra time can be given to complete the project if wasn't finished within the set duration.

Within this timescale comes other milestones. By October 2019, the PID needed to be completed and approved so that the remainder of the project could be completed. It was at this time that a simplified set of requirements would be gathered, to aid the designing process of the application. These requirements would later be updated and further detailed at the beginning of and during the implementation period. The application requirements must be outlined before the start of the design at the end of October as this must be completed before mid November to avoid delaying the start of the implementation period. The core implementation period will run from mid November to the end of February. Within this time, the vast majority of the application must be developed and tested. This is to ensure there is sufficient time towards the end of the project to complete this report.

3.3 Risks

During the development of the project there are many risks that can affect the project's outcome. Three examples of more significant risks against the project include the short timescale of the project, lack of resources and potential data loss.

As mentioned prior, the duration of the project is short, just nine months. Since I am the only developer for this project, this poses a significant risk to the project since I have only a limited amount of time to work on the project because of other commitments I have. Due to the hard deadlines set, there is little chance that more time will be allocated to allow the completion of this project.

The potential impact then, is that the project will not be completed at the end of allotted time, resulting in having unfinished work. In order to mitigate the risk of the timescale, I will have to ensure that my time is managed correctly to accommodate these other commitments and this project. This time management involves planning ahead and dedicating set times to work on the project. This ensures that my time is used at its most effective.

A threat to the project is the potential lack of resources available to myself that can allocate to the completion of the project. There is no monetary budget for this project, I will need to source resources that are available to access at no cost to myself. The number of devices I can test the application on is also limited, I have only my personal phone to test the application physically. I will have to make use of emulators to test the application at different resolutions and versions of Android. Any software needed to complete the project will need to be available for free or be accessible through my Universities systems. Being unable to source software that is required will result in a delay on working on the project. With such a short timescale with little chance of being extended, a delay cannot be afforded. Researching into the software I may need to complete this project then, is paramount. I will need to ensure that I have researched what software I need and whether it is available for me to use. If I find suitable software that is not available I must procure an alternative or plan to approach the project differently.

A significant risk to the project is the potential of data loss. Losing data hinders the progress of the project and puts pressure on the project to be completed in an already restricted timescale. To mitigate this risk I am making sure I make cloud backups of the report and code by using both Overleaf and GitHub. I also have access to two personal computers which both contain a downloaded copy of the project's code base. In the event of a failure of one PC and the loss of access to the cloud copy, I will still have a local copy to work from.

4 Requirements

This chapter covers the requirements of the application. In order for the application to be considered successful then it must meet certain requirements. Section 4.1 covers how the requirements were gathered, Section 4.2 details the functional requirements whilst Section 4.3 describes the non-functional requirements.

4.1 Requirements Gathering

Most of the requirements were conceptualised during the creation of the Project Initiation Document (PID). This was done to aid in the designing of the application. These initial concepts were formed through discussion, an understanding of what is expected from the application and research into existing systems. Throughout the course of the design period, these requirements were altered to suit new ideas and received feedback. The requirements below represent the final versions as of the end of the design period.

4.2 Functional Requirements

The following are the functional requirements of the project. These have been categorised into the 3 main categories. These are Must Do's, Should Do's and Could Do's. All requirements in the Must Do's category are the bare minimum that the application should accomplish. Given the iterative nature of the project, these were planned to be completed first with feedback being given as they were being completed. The Should Do's requirements represent what the application should realistically be able to achieve by the end of the implementation period. Finally, the Could Do's represent what could be achieved if the implementation period has any extra time once the other two categories have had their requirements fulfilled. The reasoning behind this categorisation is to aid in the implementation period by focusing on the functionality that is imperative to the success of the application and reserving potentially optional requirements for later.

Must Have Requirements

MFR1	**The Application must be available for Android Devices.**
The app must be developed for use on Android phones. This requirement is in place as there is no time in the implementation period to develop for both iOS and Android devices.	

MFR2	**Problems must be presented to the user.**
A problem including all components must be displayed to the user. These components include the blocks, slots, algorithm and a description of the problem.	

MFR3	**Blocks must contain a segment of code.**
Blocks must contain a segment of code each. This segment of code must be a portion of the solution or, optionally, be a distractor that is not included in the solution.	

MFR4	**Problems must include at least four blocks.**
Each problem must include at minimum four blocks. At least two of these blocks must be required to create a solution to the problem. Blocks not included in the correct solution are distractors, these can be used optionally to increase the difficulty of a problem.	

MFR5	**Users must be able to drag and drop answer blocks into slots.**
Using the touch screen of their device, the user must be able to drag a block from its starting position and then place it into a slot of the algorithm. The slot should then be updated, showing the code that was present in the dragged block. The dragged block must then not be able to be dragged again from its original position.	

MFR6	**Algorithms must include at least two slots.**
An algorithm within a problem must include at minimum two slots that can have blocks dropped into.	

MFR7	**The user must be able to drag blocks currently filling a slot and drop them into a separate slot.**
The user must be able to drag a block that is occupying a slot and drop it into another slot. The first slot must then be emptied.	

MFR8	**The user must be able to drop a block into an already occupied slot.**
The user must be able to drag a block and drop it into a slot that is already occupied by another block. The slot must then be filled with the new block and the previous block must be returned to its original position and able to be dragged again.	

MFR9	**Upon submitting their answer, the user should receive feedback in the form of whether they got the answer correct or not.**
After selecting to submit their answer, the application should check the given solution and check it against the stored solution. If the stored and given solutions are the same, the user has correctly answered the problem and the application should feedback this to the user. If the two solutions are not the same, the users has incorrectly answered the problem and the application should then let the user know of this.	

MFR10	**If the answer the user gave is correct, the next problem must be loaded.**
If the user has submitted a correct solution, after letting their user know of their correct answer, the application should then load the next problem. The next problem should simply be the next indexed problem.	

Should Have Requirements

SFR1	**Allow the user to select what topic of questions they would like to answer.**
The user should be able to select a series problems to answer based on their topic. The topics should include: "Ifs", "Loops" or "All Problems". All problems in the "Ifs" category must include at least one if statement, all problems in the "Loops" category should include at least one loop and "All Problems" should be include all problems present in the application. The order in which the problems are given to the user should be the problems index number in ascending order.	

SFR2	**The application must be able to read all questions from a file.**
Problems present in the application should be read from a separate file and not be hard-coded into the application.	

SFR3	**While answering a problem, the user should be able to skip the problem and a new problem should be loaded.**
At any point while answering a problem, the user should be able to choose to skip the problem and load the next problem in the series. If a problem is skipped, the current score for that problem should not be recorded.	

SFR4	**Upon submitting their answer, if the answer is incorrect, the user should receive a hint.**
If the user's answer is incorrect, they should be automatically be given a hint. However, the user should only be able to receive at maximum one hint per problem. The hint should reveal the correct location of a block in the correct solution by filling the slot with the correct block then disabling that slot.	

SFR5	**When the user submits their answer, they should receive a score based on the time it had taken them to get to that answer.**
Upon submitting a correct answer, the user should receive a score. This score should correlate to how quickly it had taken the user to complete the problem. The time taken should be measured between the problem being given to the user and the user submitting a correct answer.	

SFR6	**All user data should be saved onto and subsequently loaded from the host device.**
All data related to the user's scores for each problem should be saved to their device. If the user closes the app then opens it at a later date, their scores should be loaded into the app.	

Could Have Requirements

CFR1	**Questions given to the user must be suited to them as an individual.**
Problems presented the user should be selected based on the user's previous activity with the app. Problems should be selected based on their difficulty and the scores the user has achieved in certain topics. If the user has scored low when answering problems of a particular topic, easier problems of that topic should be selected for the user and vice versa.	

CFR2	**The user must be able to see their overall score from all of the problems they have correctly answered.**
The users' total score must be totalled and displayed somewhere in the app.	

CFR3	**The app should be able to record topics that the user has repeatedly score highly in.**
The app must record an average mark from all the scores saved for problems of all topics. From this average, the app must determine topics that the user has a high average in and topics that a user has scored lower in.	

CFR4	**All problems should be tagged on their difficulty.**
All problems present in the app should be tagged on their difficulty. A problems difficulty is decided on the number of blocks and distractors present and the number of topics they are categorised in. A problem with eight blocks including three distractors that also contains a loop and a conditional statement in it's algorithm, for example, should be considered a more difficult problem.	

CFR4	**On submitting a correct solution, the user be able to choose to attempt a harder problem of the same topic as the previous.**
Upon submitting a correct solution, the user should be given the option to be given a problem of a harder difficulty in the same topic as the problem just answered. The alternative option is to continue answering problems of the currently selected series.	

4.3 Non-Functional Requirements

The following are key non-functional requirements directly related to this project that must be considered when designing and implementing the app.

4.3.1 Usability

The drag and drop functionality of the application should feel natural to a user. As it is the main form of interaction between the user and application when completing a problem it should require minimal effort to action. To accomplish this "natural" feeling the application should give instant feedback to the user that they have completed the dragging and dropping action. This feedback should be either a graphical animation or a simple sound effect to demonstrate the action is complete. The drag and drop feature should also have a method to indicate the drag action has started. The block being dragged should follow the users finger as they drag it across the screen. This would also show to the user what block they had dragged, in the case they may have dragged the wrong one. Initiating the dragging action should also be easy for the user. The bounds around the block in which the user has to select in order to begin a drag should be at least the same size as the block itself. This area could also be larger than the block but should not overlap other blocks to prevent confusion.

4.3.2 Supportability

The application should be developed for the maximum number of potential user's possible. Provided there is an efficient enough work-around that doesn't affect the functionality of the application, code libraries that requires a minimum version of Android should be avoided.

4.3.3 Robustness

The drag and drop functionality should be able to handle all types of actions required and possess the required error handling to be able to prevent unexpected interactions from users. In the case that an unexpected action occurs, the app should be able to process the action without crashing.

5 Design

This chapter discusses the designing process undertaken to create the initial plan for the app and justification for many of the decisions made in order for the app to meet its requirements. Section 5.1 describes the navigation that needs to be implemented to allow the user to access each screen and their functionality. Section 5.2 delves into further detail about each screen, specifically how each screen's layout was designed with justification while Section 5.3 discusses the colour scheme and font chosen to create the appearance of the app. Finally, Section 5.4 describes the functionality required for the drag and drop feature present to allow the user to complete problems and Section 5.5 explains the reasoning behind the designs for the problems themselves.

5.1 Navigation

The first phase of the designing the app is identifying what screens should be present. Following the requirements, three screens should be present within the app, the *Home Screen*, *Problem Screen* and the *Level Select Screen*. I decided on having just three screens with the intention to keep navigation across the app as simple as possible.

Figure 2 shows the proposed navigation between each of the screens. From the Home Screen, the user should be able to select either to start answering any problem or to choose a particular problem. In the case that they wish to start answering any problem, they continue to the Problem Screen. If they wish to choose a problem, or series of problems, they continue to the Level Select Screen. From the Problem Screen, the user should be able to return to the Home Screen and cancel out of completing the current problem. Within the Level Select Screen, the user should be able to return to the Home Screen or by selecting a problem or topic be taken to the Problem Screen to start answering problems. There is no way to navigate to the Level Select Screen from the Problem Screen without firstly navigating to the Home Screen. This is intentional in order to reduce the number of buttons on the Problem Screen. The need to navigate to another screen, I believe, is a suitable compromise to reduce the complexity that already exists within the Problem Screen.

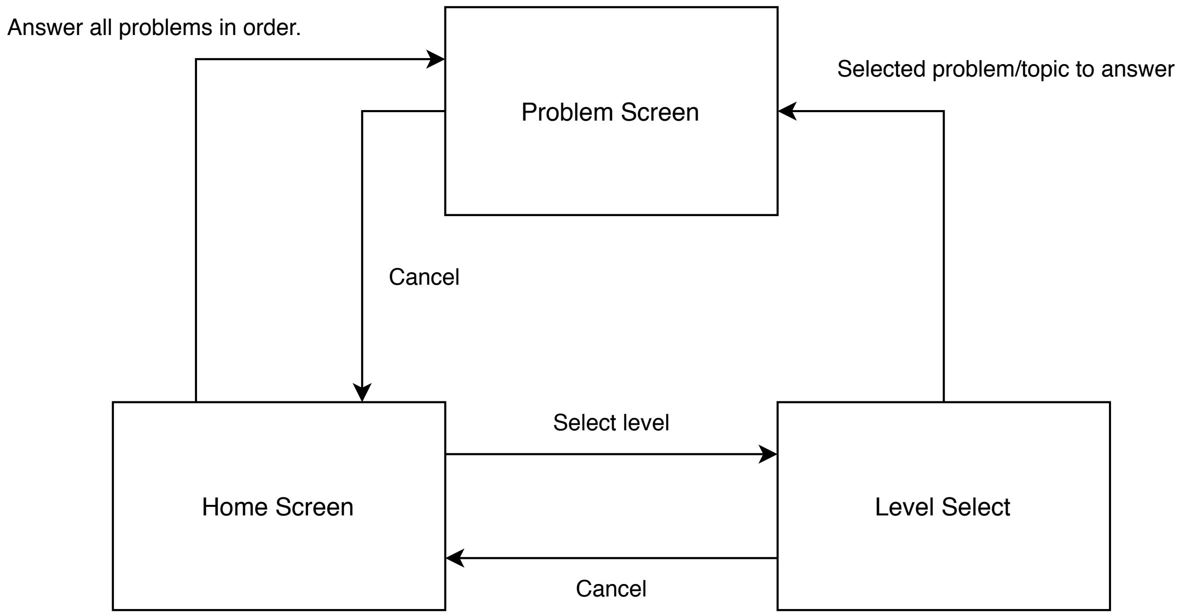

Answer all problems in order.
Problem Screen
Selected problem/topic to answer
Cancel
Select level
Home Screen
Level Select
Cancel

Figure 2: Navigation Design

5.2 Layout Design

As mentioned in the Navigation section, there are three main screens: the Problem Screen, the Home Screen and the Level Select Screen. The most important of these screens is the Problem Screen due to it containing the core functionality of the app and being the screen that the users will interact with for the most amount of time. Due to its importance, this screen was the first to have its layout finalised. Any design decisions made for the Problem Screen would be transferred over to the other screens where necessary to introduce consistency across the app's design.

Below is each of the screens' layout with justification for decisions made during their design. For the Problem Screen, a mock-up was made first, with feedback sought after on how to improve its design. After receiving feedback, a second mock-up would be made with changes made reflecting the feedback given. The other two screens each had mock-ups made but feedback was given later in the implementation phase as they were considered non-critical compared to the Problem Screen. Each mock-up focuses purely on the layout of the screen, its appearance would be decided later on in the implementation phase. At this point, the location and size of any buttons or other objects such as blocks and slots must be decided first.

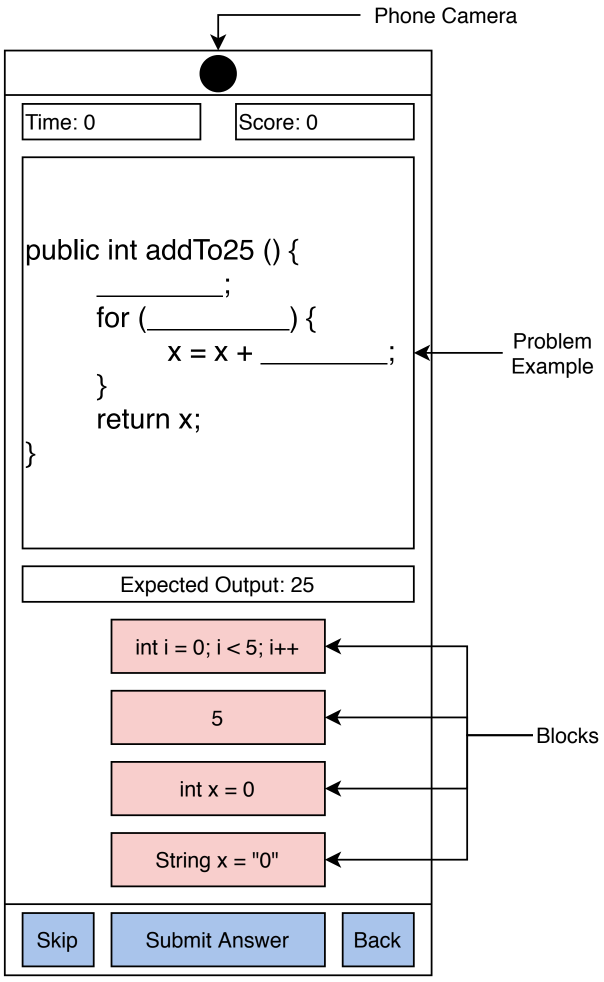
5.2.1 Problem Screen Layout

The Problem Screen contains the most user interaction and functions critical to the apps' success. It is for this reason that its layout must be thoroughly planned, reviewed and baselines ahead of the implementation phase. Any re-designs needed later on in the projects life-cycle would potentially delay the apps' completion.

The layout of the Problem Screen must include the following points in its final design:

- It must facilitate all the functionality related to solving problems.
- The user's focus must be on the problem displayed.
- The layout must be simple to understand and not require any additional instruction.
- It must contain all buttons needed for navigation.

To provide context, blocks are intended to be objects that can be dragged across the screen by the user and each contains a small piece of code. Slots are blank spaces within the problems' algorithm, blocks must be able to be dragged into these spaces. The mock-up for this screen must include both of these objects in order to provide the required functionality.



```
public int addTo25 () {
        _________;
        for (__________) {
                x = x + _________;
        }
        return x;
}
```

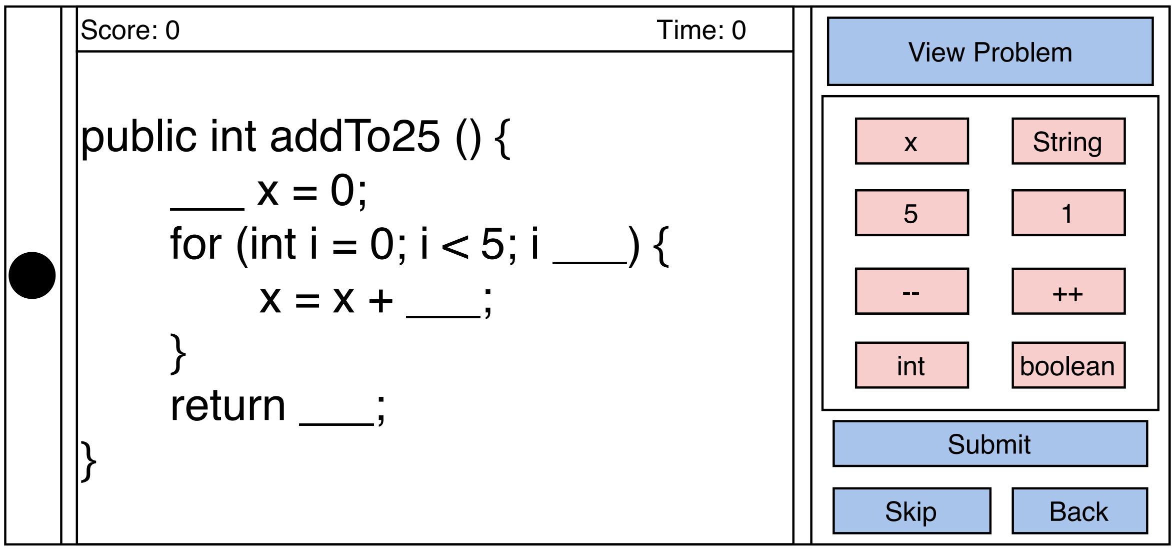
Figure 3: Portrait Design

Figure 3 represents the first design for the Problem Screen layout. As shown, this layout displays the problem at the top of the screen with the buttons and blocks at the bottom. This layout was decided on for the following reasons.

- The blocks and buttons being located at the bottom are easier to reach for the user when operating the device with one hand.
- The problem should be the focus of the user's attention, so it was positioned near the top and sized to be the largest part of the interface.
- The blocks were placed close to the problem to shorten the distance the user would

have to drag the blocks into place. The expected output is used to provide a clear separation between the blocks and the problem to prevent confusion between the two.

Once this mock-up was complete, feedback was garnered from the project's supervisor. The general concern was that this layout was too restricting. The narrow width of the area reserved for displaying the problem's algorithm would cause issues where it contained longer lines of code. Long lines of code are easily found, especially taking into account indentation which is necessary to read and understand the algorithm. The number of blocks being limited to four restricted the potential of each problem, especially if there were to exist distractors. To solve the problem of the narrow interface, the layout would be altered to conform to a landscape orientation instead. In a report by ScientiaMobile in 2014 (ScientiaMobile, 2014), they found that 94% of users visiting websites they were recording data from, had their phone in the portrait orientation. Despite wishing to conform to this trend, a compromise had to be made for the sake of the application working as intended and displaying problems correctly.

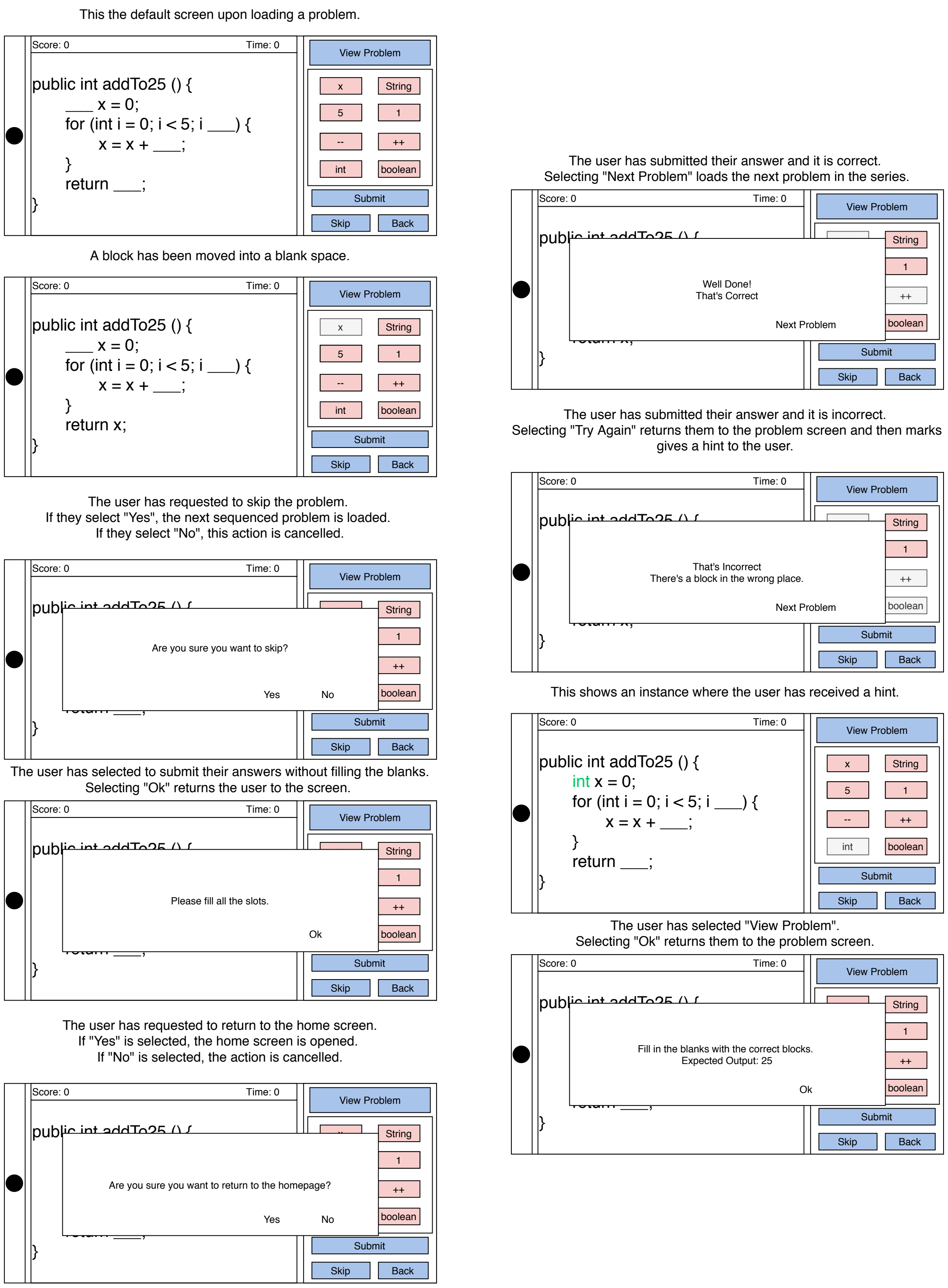


```
public int addTo25 () {
        ____ x = 0;
        for (int i = 0; i < 5; i ____) {
                x = x + ____;
        }
        return ____;
}
```

Figure 4: Landscape Design

Figure 4 shows the re-design following received feedback and some more consideration into what is expected from the problems to be solved. As shown, the problem is displayed on the left with the buttons and blocks on the right. This change in screen orientation has dramatic changes the layout of the app but has no effect on how the app is meant to be used. Not all of these changes are improvements however. The change to landscape means that the device has to be operated with both hands which could be considered more cumbersome compared to using it in portrait mode. It also introduces the problem that an algorithm is now limited in the number of lines it can have. However, this change has the overall benefit that a problem's algorithms are not restricted in terms of line length, increasing the number of potential problems that could exist in the system. It also has the added benefit of reducing the risk of requiring a horizontal scroll in order to be able to read the longer lines. Overall, I believe this change in orientation is a suitable compromise.

Among the changes in layout, the number of blocks was increased from four to eight. Having at most four blocks restricted the possible difficulty of a problem as having more blocks allows for more distractors that can be added to increase its difficulty.



This the default screen upon loading a problem.

```
public int addTo25 () {
    ___ x = 0;
    for (int i = 0; i < 5; i ___) {
        x = x + ___;
    }
    return ___;
}
```

A block has been moved into a blank space.

```
public int addTo25 () {
    ___ x = 0;
    for (int i = 0; i < 5; i ___) {
        x = x + ___;
    }
    return x;
}
```

The user has requested to skip the problem.
If they select "Yes", the next sequenced problem is loaded.
If they select "No", this action is cancelled.

Are you sure you want to skip?

The user has selected to submit their answers without filling the blanks.
Selecting "Ok" returns the user to the screen.

Please fill all the slots.

The user has requested to return to the home screen.
If "Yes" is selected, the home screen is opened.
If "No" is selected, the action is cancelled.

Are you sure you want to return to the homepage?

The user has submitted their answer and it is correct.
Selecting "Next Problem" loads the next problem in the series.

Well Done!
That's Correct

The user has submitted their answer and it is incorrect.
Selecting "Try Again" returns them to the problem screen and then marks gives a hint to the user.

That's Incorrect
There's a block in the wrong place.

This shows an instance where the user has received a hint.

```
public int addTo25 () {
    int x = 0;
    for (int i = 0; i < 5; i ___) {
        x = x + ___;
    }
    return ___;
}
```

The user has selected "View Problem".
Selecting "Ok" returns them to the problem screen.

Fill in the blanks with the correct blocks.
Expected Output: 25

Figure 5: Landscape Design with Storyboard

Figure 5 shows the final layout design including explanations of each of the buttons functionality and how the layout will react to changes made by the user's interactions with the layout.

5.2.2 Home Screen

Compared to the Problem Screen, the Home Screen is much more simplistic given it is used purely for navigational purposes.

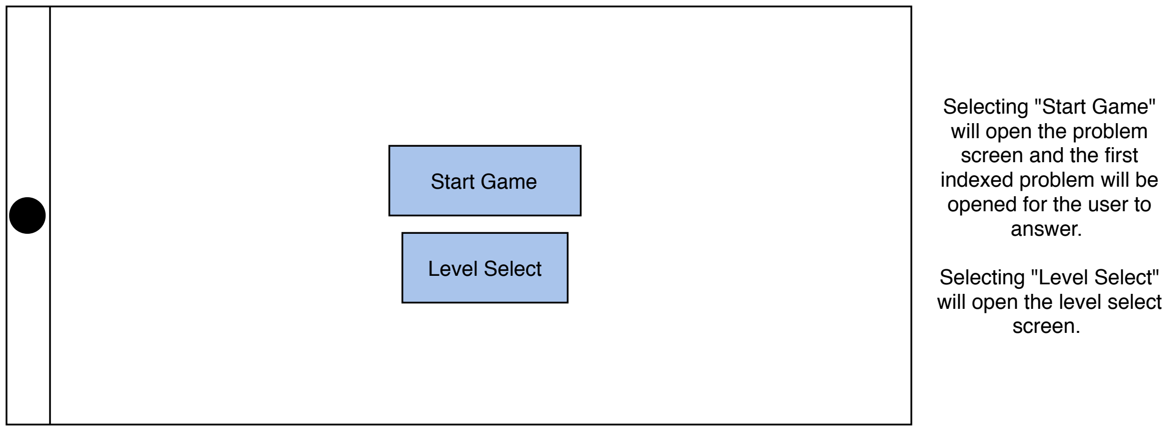

Start Game
Level Select
Selecting "Start Game" will open the problem screen and the first indexed problem will be opened for the user to answer.
Selecting "Level Select" will open the level select screen.

Figure 6: Home Screen layout.

Figure 6 shows the final design for this screen's layout. It features two buttons in the centre, "Start Game" opens the Problem Screen and "Select Level" opens the Level Select Screen. As mentioned, this screen has no other purpose other than navigation and is the first screen that will be displayed to the user when starting the app.

5.2.3 Level Select Screen

The Level Select Screen is where the user will either choose to answer a particular problem or choose to play a series of problems centred on a selected topic.

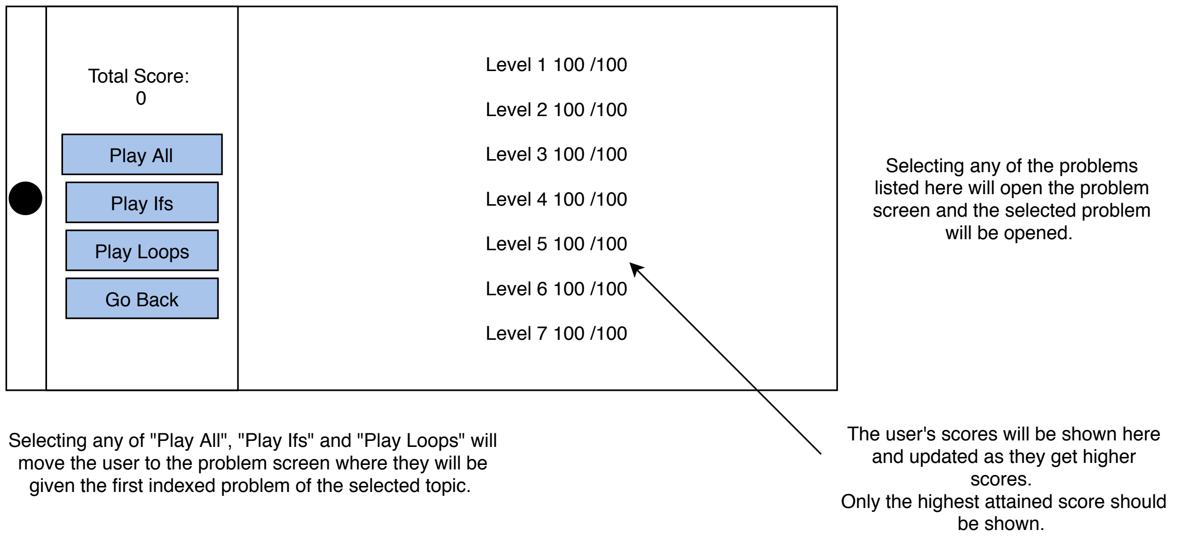

Total Score:
0
Play All
Play Ifs
Play Loops
Go Back
Level 1 100 /100
Level 2 100 /100
Level 3 100 /100
Level 4 100 /100
Level 5 100 /100
Level 6 100 /100
Level 7 100 /100
Selecting any of the problems listed here will open the problem screen and the selected problem will be opened.
Selecting any of "Play All", "Play Ifs" and "Play Loops" will move the user to the problem screen where they will be given the first indexed problem of the selected topic.
The user's scores will be shown here and updated as they get higher scores.
Only the highest attained score should be shown.

Figure 7: Level Select Screen layout.

Figure 7 shows the final design for the Level Select Screen. Similar to the Home Screen, this layout is intentionally simple and possesses little functionality outside constructing a series of problems and opening the Problem Screen. All the buttons that construct these series are displayed together along the left-hand side. There is enough blank space to allow the inclusion of further buttons that include more topics if this functionality is ever expanded on in the future. It also contains the total score that the user has achieved in their history of using the app. Each problem listed on the right side links to the Problem Screen where that problem will be displayed. As high scores are recorded, they will appear next to these links. The total score attained by the user is totalled every time the Level Select Screen and is displayed on the left-hand side.

5.3 Appearance

Later on in the development cycle, once the layouts had been established and developed, the overall appearance had to be decided upon. This involved deciding on the app's colour scheme and font.

For the colour scheme, I decided to take inspiration from the commonly known and popular "Dark Theme" that is widely present in many software IDEs. The main properties of this colour scheme is a dark background with brightly coloured portions of code interspersed in white text. The benefit of using this theme is that it is easy for users to distinguish the coloured portions of text from the white. In its use in software IDEs, it is used to highlight important sections of code such as variable or function names. This ties into my need for the user to be able to distinguish between slots, blocks and buttons in my application at a glance while also being relatable and recognisable to the app's user base.

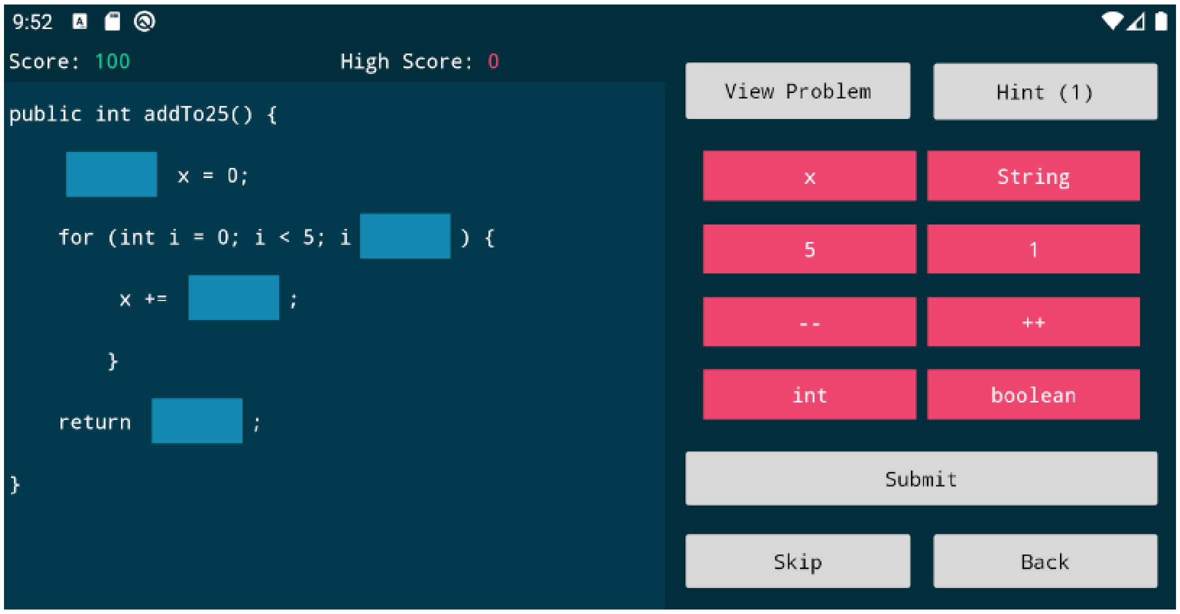

```
Score: 100                High Score: 0
public int addTo25() {
        x = 0;
    for (int i = 0; i < 5; i      ) {
        x +=      ;
    }
    return      ;
}
```

Figure 8: Final colour scheme and font face.

Figure 8 demonstrates the Problem Screen with the colour scheme I decided on. It incorporates a dark background with brighter colours used to differentiate the blocks, slots and buttons. While the entire background is dark, the background behind the problem's algorithm is slightly lighter to help focus the user's attention towards it. For the blocks, I decided on using a bright colour to make them easily seen by the user and thus, the easiest to distinguish. A brighter blue was used for the slots to contrast to the blocks. This was intentional as the empty slots needed to be distinct from filled slots. This is evidenced in Figure 9 when slots are filled, their background colour changes to match the block's background colour, creating this distinction. When a block has been

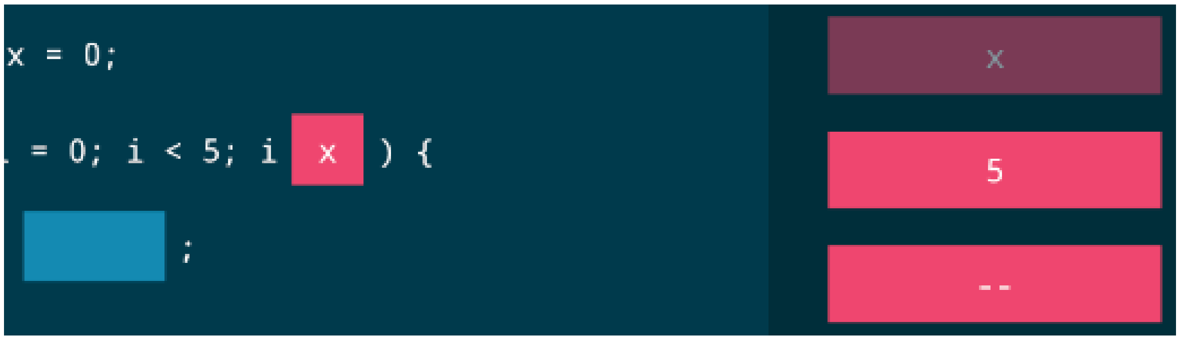


Figure 9: Blocks and slots change colour to denote status.

dragged into a slot, it is unable to be dragged from its original position until another block has been dropped into the same slot. To show this, when a block is disabled, it is greyed out. For the buttons, I decided on keeping them their neutral colour in order to keep the focus on the blocks and slots whilst not blending into the background.

For my font choice I decided on using the monospace font face. Originally I had intended only to use this for the displayed problem to make it appear more like actual code. In the final version however, I decided upon using this font face for the entirety of the application to tie further into the theme of programming and incorporate a level of consistency.

5.4 Drag and Drop

When completing a problem, the user will be making use of a drag and drop function to move blocks into slots. This function required its own set of design decisions. As mentioned in Section 4.3.1 the drag and drop needs to feel "natural" to provide a positive experience for the user. To accomplish this, the motion of dragging and dropping needed to give feedback to the user to demonstrate when they have started and stopped the action.

When starting a drag, the block being dragged should follow the users' finger but slightly offset so it is visible and not directly underneath their finger. This is a common feature in many applications that also feature dragging and dropping as it is a clear signal that a drag action has started.

A form of feedback should also be given to the user when completing a dropping action. For this, there was the option of either sound or a graphical animation to signify this. While adding a graphical animation would make the app appear more dynamic visually, it would also be resource intensive to implement since I would have to create the animation and make it scalable to different sizes of slots. Instead then, I have decided on using sound. For the sound, a simple clicking sound or snap is the most appropriate. Having a more intricate or fan-fair like sound would end up being more distracting and would ultimately undermine the reason the sound is to be included.

5.5 Problem Design

Perhaps the most important part of the app that needs designing are the problems that will be given to the user. Every problem must follow the same design decisions as these have been carefully made in order for the app to suit the requirements. These can be considered rules and are used to implement consistency throughout the all the problems.

The problems are made up of a number of components including:

- The problem algorithm.
- Blocks of code.
- Slots or blank spaces that appear in selected parts of the algorithm.
- Problem description if necessary or at minimum an expected output.
- A solution.

For every problem, the algorithm included must follow the Java syntax. Java was chosen for this as it is commonly used to teach beginners and possesses a strict syntax. This eliminates the risk that the algorithm is open to interpretation by the user and helps stop the possibility of multiple unplanned solutions being present. Each algorithm must also include just basic syntax such as loops or conditional statements since these are the topics first taught to users and should avoid using atypical libraries.

Each problem should include of at least two total blocks, with a maximum of eight. This number includes any distractors though problems can have zero distractors if the intent is for that problem to be easier. Each block can contain either a variable name, constant, operator or data type. This is to keeps blocks simple to read and understand, and ensures that if eight blocks are present, they can all fit inside the layout.

All problems should feature slots in selected positions within the algorithm. The number of slots should equal the number of blocks present in the solution, not the total number of blocks since distractors may be present. As mentioned in Section 5.3, slots will be coloured differently if they are filled or empty. This is to allow the user to clearly identify between these two states. If a slot is filled, it displays the text of the block that was dropped onto it and no text should be displayed in an empty slot.

A point of contention between myself and the project's supervisor was on the size of the blocks and slots. I felt that the size of the blocks in their original position should be the same. This presents formality between the blocks and improves the appearance of the layout. The supervisor instead believed that the better solution is that for every block, their size is scaled to the length of code they contain. The slots also, should be scaled to the length of code that is present within the correct block. I omitted including this design as I believe that this makes the solution far too obvious to the user and undermines the requirement for the user to read and understand the problem in order to create a solution.

Finally, each problem should include a short description of what is expected from the algorithm. In the case that the algorithm outputs a constant or value, this can be expressed by describing the expected output. For all other cases where there is no regular output, a description of what the algorithm is supposed to compute should then be given instead. This is simply to provide additional context to the user of what is expected from their solution.

6 Implementation

This chapter discusses the implementation phase of the project. Section 6.1 discusses changes made to the requirements during development and Section 6.2 discusses difficulties faced during the applications development. Section 6.3 describes the methodology behind how the app was tested during development and section 6.4 includes a testing table which shows the results of testing the core functionality of the app.

6.1 Requirements Changes

The following are three key examples of situations where the requirements for the app had to be altered to suit received feedback and new ideas. In all three cases, I believe that these changes have improved the final prototype over the original designs.

6.1.1 Hint System

The requirements address the inclusion of a hint system to be present within the Problem Screen. This hint system entailed revealing the position of a correct block within the algorithm in the case that the user submitted an incorrect solution. This was initially implemented into the system but following testing and feedback this was later altered.

Feedback received described that the hint was less of a hint and more the system simply completing a section of the problem for the user. This system had additional problems in that if, for example, the given problem had just two slots available, giving a hint made the problem far too easy. It also had no effect on the score, there was nothing to dissuade the user from purposefully submitting an incorrect solution just to make the problem easier.

To fix these problems, the following was implemented. Instead of giving a hint when submitting an incorrect solution, the user could instead request a hint at any time by selecting a new button on the Problem Screen. The label given to this button displays the number of hints available and is disabled when no hints remain. Currently, for every problem, one hint may be given. Following the changes made to the scoring system, 25 is also removed from the max score. This should ideally dissuade the user from using a hint in the first place. The hint itself was also changed from automatically filling a slot to disabling a distractor block not used in the correct solution, if such distractors exist. In the case that there are no distractors, the existing system of giving away a correct location is used instead. With these changes, the hint system has more depth outside simply making the problem easier.

6.1.2 Score System

As stated in the requirements, the app required to give the user a score after submitting a correct answer. The original requirement states the score given the user would be calculated from the time taken to answer the problem. After deliberation, this method of calculating score was changed.

Every time the user is now given a problem, they start at the maximum score of 100 For every hint they use, 25 is deducted. If the user submits an incorrect solution, 10 is deducted. The number of blocks the user can move without a deduction in score is calculated for every problem. This number equals the number of blocks in the correct solution. Any blocks that are moved after the using these free moves deducts 10. This means that if the user drops the correct blocks into the correct slots without any additional moves or using any hints, they can achieve the maximum score.

These changes were implemented in order to allow the the addition of consequences to using hints following the changes to the hint system. Also, it is now more explicit to the user how to achieve the maximum score and avoids having to store the margins of time that grant particular scores for every problem.

6.1.3 Feedback on Incorrect Answers

During the implementation period, I had the chance to receive some feedback on the project's progress from the project moderator. The feedback that was received asked what this app did differently over other software with a similar approach to teaching a subject using drag and drop blocks. After some deliberation a new feature was devised and added to the Problem Screen.

Initially the requirements stated that if the user submits an incorrect solution, the Problem Screen gives feedback on whether they were correct or incorrect. In the case that the user is incorrect, the screen displays the message similar to “Incorrect, Try again”. This new feature creates a more specific message on what has caused the answer to be incorrect. For example, if the wrong data type was used, the screen now shows “Incorrect. You have a compilation error, are you using the correct data types?”.

This new feature gives further depth to the feedback given to the user and is aimed at helping them understand why their solution is incorrect. As it stands, this new feature is in its prototype form as it was added late into the implementation phase. In the future, I hope to expand on this feature and include more specific error messaging related more to each problem individually.

6.2 Implementation Difficulties

The following section describes key difficulties I faced when implementing the app and how I overcame them to produce the final version.

6.2.1 Relative and Linear Layouts

Upon loading the Problem Screen, the screen has to construct the problem’s algorithm. The algorithm is constructed from text boxes to contain code and slots. In the first version of this function, I made use of relative layouts to contain the algorithm being displayed. Relative layouts require any objects being added to itself to declare where they should be located compared to other objects already present within the layout. For example, a new object could be added to the layout with the properties that it has to be positioned above and to the right of another specified object. This layout was chosen as I believed it to be the optimum layout for organising the algorithm. Each text box and slot had to be positioned in such a way that the algorithm resembled real code and was easily readable by the user.

This layout had issues however. It is unknown to me what the root cause of the issue was or whether it was within my control but following a software update on my personal phone which I was using to test and demo the app, the layout ceased to work as intended. The layout would continuously overlap all text boxes and slots on top of each other in the top left hand corner when the Problem Screen first loaded. Dragging a block to one of the slots fixed the layout but this issue on initial start up was inexcusable to leave without fixing. Despite attempting multiple fixes, I could not get the layout to work correctly for my device. I instead had to dismantle the relative layout in favour for another type.

The alternative to using relative layouts is using linear layouts. The layouts are simplistic in how they work. As objects are added to the layout, they are automatically positioned either to the right of the last object or below, depending on whether a horizontal or vertical layout is used. I used a combination of the two together to create the new layout needed to correctly represent the algorithm as shown in Figure 10.

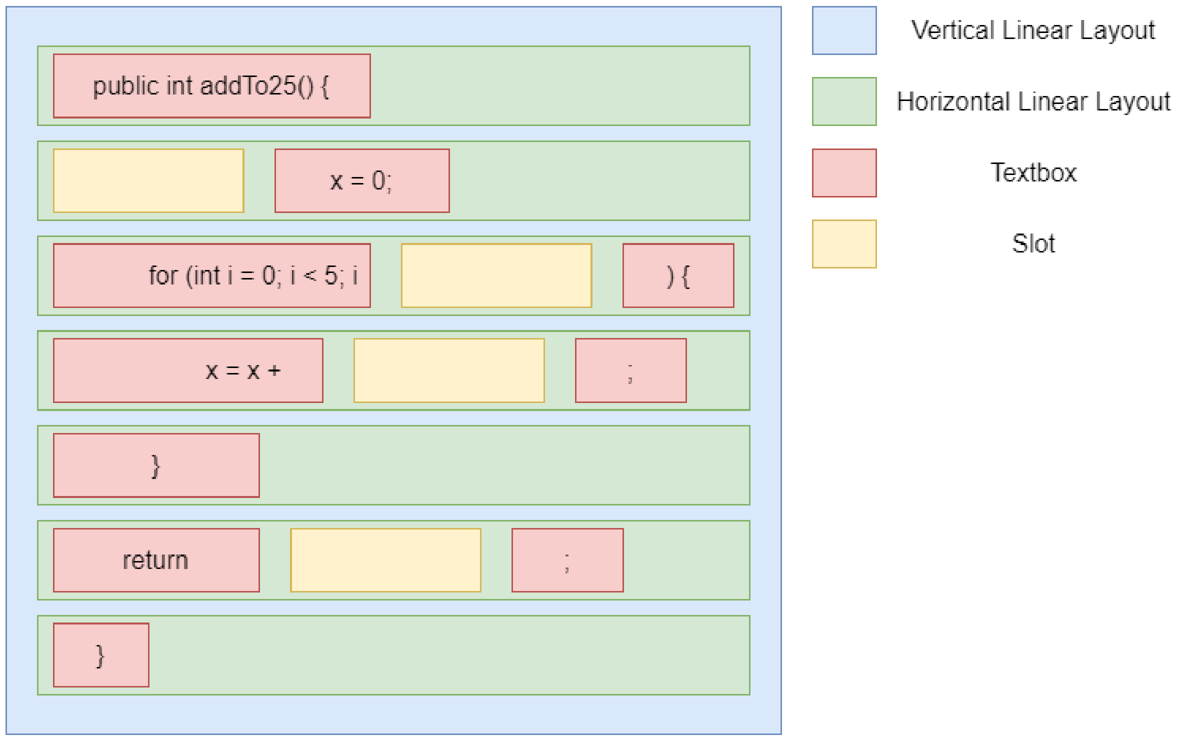


Figure 10: Linear Layout Design (Not as depicted in actual app)

Though this solution requires more objects to be present in the layout and uses more resources from the host device, this was a simpler solution over spending more time attempting to fix the previous.

6.2.2 Drag and Drop

In order to meet requirements, the drag and drop functionality had to meet certain criteria.

In order to meet the robustness that the app requires, I had to make use of many edge cases to compensate for any actions that are prohibited or unexpected. Since there can exist multiple locations that blocks can be dropped into or dragged from, the action listeners used to provide this functionality had to be implemented in such a way that they were able to distinguish between block and slot. In the case that an unexpected action occurs, the action listeners are designed to compensate and avoid crashing.

As mentioned prior in Section 5.4, the drag and drop function required a form of feedback for when the user dropped a block into position. After researching online, I found a sound available under the creative commons license that suits the requirements for suitable feedback (Topschool, 2018). This sound is preloaded on start up of the Problem Screen and plays whenever a drop action is performed.

An additional feature was added to improve the drag and drop that wasn't featured in the initial design. This feature is to allow the user to return a block to its original position. To implement this feature I added a transparent text box on top of the block's layout. This text box was given an action listener that, when a block is dropped into it from a slot, clears the slot and enables the block to be dragged again from its original position.

6.3 Testing Methodology

The vast majority of testing for the app was completed during the implementation phase of the project. As functions were being completed, they were being tested on my personal devices or emulators. If a bug was found, a fix was implemented before the next function would be developed. This method relates to the requirements categories. Requirements in one category had to be completed to a satisfactory level before moving onto the next category. In order to be considered complete, they had to be functional and tested enough that I was confident they were working correctly.

However, even with testing during development, some bugs can still be present in interactions between new components added to the app. It is for this reason that I have created a short testing table that includes tests for all critical functions of the app.

6.4 Testing Table

The following is a testing table which describes tests that I completed against the app and whether the test resulted in a pass of fail. Since I will be testing core functionality, I am expecting most if not all tests to result in a pass. This is also due to testing already having taken place during development.

Test Description	**Pass/Fail**
From the homepage, open the Problem Screen by select “Start Game”	**Pass**
From the homepage, open the Level Select Screen by selecting “Select Level“	**Pass**
From the homepage, open the Level Select Screen by selecting “Select Level”	**Pass**
Selecting a level from the Level Select Screen opens the Problem Screen and the selected problem is shown	**Pass**
Selecting the “Play Ifs” button opens the Problem Screen and the only problems displayed possess at least one if-statement.	**Pass**
Selecting the “Play Loops” opens the Problem Screen and the only problems displayed possess at least one loop.	**Pass**
Selecting a the “Play All” topic opens the Problem Screen.	**Pass**
Dragging a block from its original position over a slot “fills” the slot. The slots background colour changes to match the blocks and the slot now contains the same text as the block. The block that was dragged has it’s background colour changed and is unable to be dragged.	**Pass**
Dragging a block from a slot to another slot “fills” the slot. The first slot is then “emptied”. It reverts to its original colour and is cleared of all text.	**Pass**

Continuation of Testing Table	
Test Description	**Pass/Fail**
Dragging a block from a slot to another slot that is already fills overwrites its contents. The first slot is then “emptied”. It reverts to its original colour and is cleared of all text. The overwritten slot now contains the text that was present in the first slot. The block that was used to fill the overwritten slot reverts is background colour to its original and is now able to be dragged again.	**Pass**
Selecting “View Problem” on the Problem Screen creates a new dialog window that displays the description of the problem being completed. Selecting “Dismiss” closes this dialog box.	**Pass**
Selecting “Hint 1” causes a distractor block to change its background colour and is no longer draggable. The “Hint 1” button should have its text changed to “Hint” and be disabled.	**Pass**
Selecting “Skip” on the Problem Screen then “Yes” on the opened dialog box refreshes the Problem Screen and opens the next problem in the given series.	**Pass**
Selecting “Back” on the Problem Screen then “Yes” on the opened dialog box opens the Home Screen.	**Pass**
Selecting “Skip” on the Problem Screen then “No” on the opened dialog box closes it and no further action is taken.	**Pass**
Selecting “Skip” on the Problem Screen then “Yes” on the opened dialog box closes it and no further action is taken.	**Pass**

The results of the tests show that the core functionality of the app is working as intended. I do not believe, however, that the app is free of all bugs. If time was more permitting, further testing into the interfaces between components would be conducted.

7 Evaluation

This chapter consists of evaluation of the final version of the app against the specified requirements found in Chapter 4. Section 7.1 compares the functionality within the app and the functional requirements specified in Section 4.2. Section 7.2 reflects on the included functionality that fulfils the non-functional requirements.

7.1 Evaluation against Functional Requirements

This section analyses each requirement and gives a summary of whether that requirement was fulfilled or not. In the cases that a requirement has been fulfilled, to what extent the requirement has been fulfilled is described. If a requirement has not been completed, possible methods of implementation or discussion into changes to this requirement will be suggested.

Must Have Requirements

MFR1	**The Application must be available for Android Devices.**
Fully Met	
The app was developed in Android Studio, an IDE that specialises in developing Android apps. The app has been testing on a physical Android device and emulator.	

MFR2	**Problems must be presented to the user.**
Fully Met	
The Problem Screen is used to display a problem and all of its components. See Section 5.2.1 for the layout designs for this screen.	

MFR3	**Blocks must contain a segment of code.**
Fully Met	
Blocks contain either a segment of code that is of either a variable name, constant, operator or data type.	

MFR4	**Problems must include at least four blocks.**
Fully Met	
Every present problem contains at least four blocks with a maximum of eight. The number of blocks is dependent on the problem.	

MFR5	**The user must be able to "drag and drop" answer blocks into the blanks sections, or slots, of the given algorithm.**
Fully Met	
The user is able to drag a block from the right hand side of the Problem Screen and drop it into any slot on the algorithm.	

MFR6	**Algorithms must include at least two slots.**
Fully Met	
Every problem present in the app has at least two slots.	

MFR7	**The user must be able to drag blocks currently filling a slot and drop them into a separate slot.**
Fully Met	
The user can move blocks between slots and can also return block to their original position.	

MFR8	**The user must be able to drop a block into an already occupied slot.**
Fully Met	
The user is able to drag a block from one slot onto another, even if the new slot is already occupied.	

MFR9	**Upon submitting their answer, the user should receive feedback in the form of whether they got the answer correct or not.**
Fully Met	
The user is told if there answer is correct or not in a dialog box that is opened when they submit their answer.	

MFR10	**If the answer the user gave is correct, the next problem must be loaded.**
Fully Met	
If the user's answer is correct, upon closing the opened dialog box, the next problem is loaded.	

Should Have Requirements

SFR1	**Allow the user to select what topic of questions they would like to answer.**
Fully Met	
Through the Level Select Screen, the user is able to select the ”If's” and ”Loops” topics. They can also select to play all problems.	

SFR2	**The application must be able to read all questions from a file.**
Fully Met	
On startup, the app reads all problems from an asset file and parses them into Problem objects.	

SFR3	**While answering a problem, the user should be able to skip the problem and a new problem should be loaded.**
Not Met	
From the Problem Screen, the user is able to select the ”Skip” button. This opens a dialog box that requests confirmation from the user. Upon selecting ”Yes” to confirm, the next problem in the series is loaded.	

SFR4	**Upon submitting their answer, if the answer is incorrect, the user should receive a hint.**
Changed Requirement	
This requirement was altered as mentioned in Section 6.1.1. In the final version of the app, no hint is given on submitting an incorrect answer for the first time, however the user has the option to request a hint from the Problem Screen.	

SFR5	**When the user submits their answer, they should receive a score based on the time it had taken them to get to that answer.**
Fully met with alterations	
When submitting an answer, the user is given a score. The method of how a final score is calculated has been altered, as mentioned in Section 6.1.2.	

SFR6	**All user data should be saved onto and subsequently loaded from the host device.**
Fully Met	
As new scores are recorded, they are immediately saved to the host device before the next problem is loaded. Upon app startup, all scores are loaded.	

Could Have Requirements

CFR1	**Questions given to the user must be suited to them as an individual.**
Not Met	
Problems presented to the user are from the series selected by themselves, either from the Home Screen or Level Select Screen. Problems delivered to the user are ordered by their index. This requirement was not fulfilled due to time constraints. If it was to be implemented, the topic of problems would still be decided by the user through the Level Select Screen. However, the app would decide whether to give an easier or harder problem depending on the average scores that the user has attained for the selected topic. This analysis would be conducted every time a new problem is required to be delivered to the Problem Screen in order to make a suggestion based on the most recent data.	

CFR2	**The user must be able to see their overall score from all of the problems they have correctly answered.**
Fully Met	
The users' total score is displayed on the Level Select Screen.	

CFR3	**The app should be able to record topics that the user has repeatedly score highly in.**
Not Met	
This requirement has not been met, the only data recorded on the user's score is their score at this time. If this was to be implemented, the average score would be calculated and recorded every time a new high score is saved.	

CFR4	**All problems should be tagged on their difficulty.**
Not Met	
The problems are not tagged on their difficulty anywhere in the app. If this was to be implemented, either the difficulty would be decided and recorded alongside the problem in the problem asset file or the difficulty would be calculated upon loading the problems.	

CFR5	**On submitting a correct solution, the user be able to choose to attempt a harder problem of the same topic as the previous.**
Not Met	
Currently, the user is given the next problem in the series to complete, they have no option to attempt a harder question. If this was to be implemented, the dialog box displayed on correctly answering the problem would also have a selectable option as to whether the user would want to attempt a harder problem. In this case that this is selected, a harder problem is requested and subsequently delivered to the Problem Screen. If no problem exists that is more difficult, a problem of the same difficulty would be delivered instead.	

7.2 Summary of Evaluation against Requirements

Overall, most of the requirements were met. The most important requirements, those in the "Must Do's" and "Should Do's" category were all fully met, though some had alterations. All but one of the requirements in the "Could Do's" category were not met, though a description of the process needed to implement these requirements was discussed for every requirement not met. This results in the app having all of its core functionality but missing some of the extra functionality that would make it more successful in it requirement to teach beginner programmers in algorithm construction. In future, these missed requirements and potentially other new features could be implemented as I believe they would do nothing but improve the app.

7.3 Evaluation against Non-Functional Requirements

This section analyses how well the app in its final version fulfills the non-functional requirements.

7.3.1 Usability

This requirement focused on the need for the drag and drop function to feel ”natural” to the user and require minimal effort to complete. In order for this the app to fulfill this need, the actions of the user needed feedback for when a drag and drop action started and ended. This was met as evidenced in Section 6.2.2 which discusses the use of sound when the user drops the block in place.

7.3.2 Supportability

This requirement specified the need to avoid using functions and libraries that required a minimum use of Android as much as possible during development. This was adhered to as in most areas but was unavoidable in the functionality required for the drag and drop actions. Due to the need to use an external library, the minimum version of Android that can be used is 6.0.1 (Marshmallow). Though this reduces the number of devices that can use this app, this version was released almost five years ago at the time of this writing. This means that the number of Android devices that are able to use this app should still be in the majority. This was a necessary compromise in order to include the core functionality of the app.

7.3.3 Robustness

During implementation of the drag and drop functionality, the action listeners used include error handling and checking edge cases in order to prevent crashes. The number of objects that make use of these action listeners is kept to a minimum by removing functionality from objects that don’t require it at the time. This is to reduce the risk of unexpected actions and interactions between objects to a minimum.

8 Conclusion

This chapters focuses on my personal reflections on the entire project. While I believe that the project as a whole was a success, there are areas that require improvement, both in myself and the final prototype.

Project Successes

App development throughout the implementation phase had few major issues that delayed the app from being complete. The only exceptions to this were the layouts needed for the algorithm on the Problem Screen as mentioned in Section 6.2.1, as well as passing objects between activities. These required more research into solving since I had no prior experience in developing mobile apps for Android. Most feedback given was implemented quickly and helped solve many problems that the app had during development and the final product is one that I am very happy with.

The projects pre-planning went very well, both the project's supervisor and I had a good idea of what to expect from the final product from the start. This made specifying the requirements and designing the app a lot easier.

Project Difficulties

At some stages of the project, my views on certain functions behaviour conflicted with the project supervisor's view. In particular was the difference in views on the size of the blocks as mentioned in Section 5.5. Our opinions differed on the app's appearance when using the drag and drop feature. This difference had an effect on the final version of the app, since I went with my own design over the project's supervisors. In future projects, I will attempt to conceptualise both ideas to more easily weigh up the pros and cons of each solution and promote further discussion on what solution is the best for the project. This was not done for this project due to time constraints and it not being a vital component needed for the app to fulfil its requirements.

Another difficulty for the project was my own shortcomings in that I would focus too often on the difficulty of implementation of new features rather than the improvements these features would provide. Going into future projects, I need to focus on the product itself and not as much on potential problems as this could restrain a product from being more successful. I believe that this had a negative impact on this project as many features in the app could be improved if I didn't concern myself so much with how difficult they could be to implement.

What I have learned

At the start of this project, I had no experience of developing a mobile app. Throughout the implementation and design phase, I had to research many topics to build up a knowledge base that would allow me to develop the app to a satisfactory degree. It was exciting to learn new concepts and frameworks and I would absolutely work on developing apps again in the future.

In terms of the software engineering side, I have learned that I should take more care into considering methodology of how I approach a project such as this one. I have also learned that I need to make sure that I can justify every design decision that I make. By justifying myself, I can prove that I have put thought into my design and it also leaves me open to critique and feedback which I should embrace rather than dismiss.

Different Approaches

If I were to complete this project again, I would spend more time on the pre-planning before moving onto the design and implementation. As it stands, I am unhappy with how vague my requirements are for this project. A more specific set of requirements that I can use to focus my development would be far more beneficial to me as an individual. I would also spend more time on the layout mock-ups but not before spending time researching on the capabilities of environment. Having no experience in creating mobile app layouts meant that I had no knowledge on what was possible when creating screen layouts. Perhaps having more knowledge on this subject would result in more exciting and dynamic app designs.

Future of the project

If there is a future for this project that would involve further development of the app, I would implement more of the requirements stated in the Could-Do's category. I would also add far more problems and expand on the number of topics that could be chosen by the user. An area that could be explored more would be how to make the app beneficial to more advanced programmers. This could include simply adding harder problems or creating a new type of problem entirely. The one feature I would include would be the inclusion of other programming languages such as Haskell. This coupled with targeting advanced programmers could vastly increase the potential user base of the app.

References

Battistella, P., & von Wangenheim, C. G. (2016). Games for teaching computing in higher education–a systematic review. *IEEE Technology and Engineering Education*, *9*(1), 8–30.

Brown, N. C., & Wilson, G. (2018). Ten quick tips for teaching programming. *PLoS computational biology*, *14*(4).

Brown, Q., Mongan, W., Kusic, D., Garbarine, E., Fromm, E., & Fontecchio, A. (2013). Computer aided instruction as a vehicle for problem solving: Scratch programming environment in the middle years classroom. *Retrieved September*, *22*(6.1), 1.

bt Rahim, H., Zaman, H. B., Ahmad, A., & Ali, N. M. (2018). Student's difficulties in learning programming.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the fourth international workshop on computing education research* (pp. 113–124).

Ericson, B. J., Margulieux, L. E., & Rick, J. (2017). Solving parsons problems versus fixing and writing code. In *Proceedings of the 17th koli calling international conference on computing education research* (pp. 20–29).

Gibson, S. (2015). Why kahoot is one of my favourite classroom tools. *URL: http://tomorrowslearners.com/why-kahoot-is-one-of-my-favourite-classroom-tools.*

Higher Education Statistics Agency. (2019). Higher education student statistics: Uk, 2017/18 - subjects studied. Retrieved July 3, 2020, from https://www.hesa.ac.uk/news/17-01-2019/sb252-higher-education-student-statistics/subjects

Kalelioglu, F., & Gülbahar, Y. (2014). The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, *13*(1), 33–50.

Keane, J. (2017). Norwegian edtech company kahoot! reaches 1 billion players, 40 million maus. Retrieved May 6, 2020, from https://tech.eu/brief/kahoot-1-billion-players/

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 1–15.

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information technologies*, *7*(1), 55–66.

Parsons, D., & Haden, P. (2006). Parson's programming puzzles: A fun and effective learning tool for first programming courses. In *Proceedings of the 8th australasian conference on computing education-volume 52* (pp. 157–163).

Rahman, M. M., & Paudel, R. (2018). Preliminary experience and learning outcomes by infusing interactive and active learning to teach an introductory programming course in python. In *Proceedings of the international conference on frontiers in education: Computer science and computer engineering (fecs)* (pp. 51–57). The Steering Committee of The World Congress in Computer Science, Computer ...

Repenning, A. (2017). Moving beyond syntax: Lessons from 20 years of blocks programing in agentsheets. *Journal of Visual Languages and Sentient Systems*, *3*(1), 68–89.

ScientiaMobile. (2014). Movr (mobile overview report) october - december 2014. Retrieved April 23, 2020, from https://data.wurfl.io/MOVR/pdf/2014_q4/MOVR_2014_q4.pdf

Topschool. (2018). High pitch click. Retrieved January 23, 2020, from https://freesound.org/people/Topschool/sounds/443282/

Wang, H. Y., Huang, I., & Hwang, G. J. (2014). Effects of an integrated scratch and project-based learning approach on the learning achievements of gifted students in computer courses. In *2014 iiai 3rd international conference on advanced applied informatics* (pp. 382–387). IEEE.

University of Portsmouth

School of Computing

Project Initiation Document

PJE40

Mobile App for Learning Basic Algorithm Construction

Authors
Jack Turner
Supervisor
Matthew Poole

October 25, 2019

1 Basic Details

Student Name	Jack Turner
Project Title	Mobile App for Learning Basic Algorithm Construction
Course	MEng Computer Science
Project Supervisor	Matthew Poole

2 Degree Suitability

This project is suitable for my degree as it provides a solution to a complex problem related to a field in computer science, in this case the education of, and requires research of published literature and existing systems. Due to the projects requirements to develop a piece of software, this project will make use of my practical skills in programming to develop this solution as well as my prior experience in how to prepare and execute a plan of my own creation.

3 Outline of the project environment and problem to be solved

My project is based upon the problem given to novice programmers to solve called Parsons Problems. This involves learners organising mixed-up lines of code into their correct order. This method of teaching programming is used to challenge and teach novice programmers but also reduce frustration when doing so. My project will be centered on a similar approach but with a different problem style. Rather than using mixed-up lines of code, I will instead be using a given algorithm with missing chunks of code. The user will then be given code blocks that they will have to then drag into the blanks to complete the algorithm. Similar to the Parsons Problems, this method of teaching's goal is to guide and challenge the user but also remove the extra jargon that may frustrate the user. This approach will be represented on a mobile app developed for the Android operating system.

4 Project aims and objectives

The overall aim of this project is to develop a mobile app to assist novice programmers the basics of programming, including the construction of algorithms and the functions of simple lines of code.

To accomplish this project, I will make sure I study existing methods of teaching programming, particularly Parsons Problems to discover their successes and their shortcomings in order to make sure that my software is successful. I will also source relevant and specific user requirements and design the software using the most suitable layouts and architecture to suit the projects needs.

5 Project deliverables

By the end of the project I will deliver both my Project Report and my final Software Solution. I will develop my app based on requirements that I have grouped into three categories. These are "Must Do's", "Should Do's" and "Could Do's". At the end of the implementation period, I expect the application should meet all of the "Must Do's" and "Should Do's" as well as a few of the "Could Do's".

5.1 Must Do's

These are the requirements that my software must meet to be successful. These include the core mechanics of how the app will function.

- The application must be available on Android devices.
- Problems must be presented to the user with blank sections that blocks can be dragged into.
- Answer blocks must be presented to the user, including detractor (or false answer) blocks if the problem includes them.
- The user must be able to "drag and drop" answer blocks into the blanks sections of the given problem.
- The user must be able to "drag and drop" answer blocks from the blank sections of the problem back into their original position.
- Upon submitting their answer, the user should receive feedback in the form of whether they got the answer correct or not.
- If the answer the user gave is correct, the next problem must be loaded.

5.2 Should Do's

These are the requirements that my app is expected to meet. These do not include core functionality but rather important features that will have a negative impact on the app if not met.

- The application must be able to read all questions from a file.
- Questions given to the user must be suited to them as an individual.
- Upon submitting their answer, the user should receive feedback on whether their answer was correct. If the answer they gave was incorrect and its their first answer attempt at that problem, they should receive a hint. The hint should take the form of highlighting a correct block.
- The application should save the topics that the user is good at and what problems they have answered.
- All problems should be tagged on their difficulty and the topic they are testing the user.
- If the answer the user gave is correct, they should be able to select to attempt to answer a harder problem of the same topic or answer a problem of a different topic.
- The application must save the highest difficulty of each topic they have answered.
- All user data should be saved locally on the host device.

5.3 Could Do's

These requirements include optional functions that would improve the final product if there are the resources to include them.

- Allow the user to select what topic of questions they would like to answer, rather than the application deciding for them.
- When the user submits their answer, they should receive a score based on the number of answer they have submitted and the time it had taken them to get to that answer. This time should be measured between them being given the problem and them submitting a correct answer.
- The user must be able to see their overall score from all of the problems they have correctly answered.
- At any time while answering the problem, the user should be able to skip the problem and a new problem should be loaded. The skipped problem should not be recorded as completed.

6 Project constraints

My main resource constraint is time. I have to make sure I manage my time effectively to make sure that I have the time to design, build and document my project. It is due to this constraint that I am limited in the scope of the project as well. This is why I have split my software requirements into the three categories, it is to define what I must accomplish as well as what I hope to accomplish in the time I am given.

I will only be able to produce the software solution for Android devices, this is due to myself not owning any devices which run iOS to test the app as well as the increase in time required to develop on another operating system.

Due to the lack of funding on this project, I will constrained to using free software to develop my application. The lack of needing specialist software shouldn't require me to need funding for software though.

7 Facilities and resources

I will be developing my solution for the Android operating system. This requires the use of an IDE (integrated development environment) that supports Java and possess an emulator to test the software during and after implementation. I will be using Java to program this solution as it is easily available for free and is one of the most common languages used to program Android apps. The IDE I will be using will be Android Studio. It is well supported by Google, the developers of Android and suits all of my needs to develop the solution successfully. This IDE is free to download but is unavailable on University computers as of time of writing. Fortunately this is easily solved as I personally possess a computer to develop the software. For version control and keeping online backups of my code, I will be using GitHub to store my code. Repositories and readily available for free and the GitHub desktop application can be downloaded on both University and personal computers. Overall all of the software I will be using is available for free download on at least my own personal computer and will require no external funding.

8 Log of risks

8.1 Time Management

Description - The project has a limited time period in which it must be completed.

Risk Impact - If my time during the project is not managed sufficiently, the project is at risk of not being complete at the time of final submission.

Mitigation / Control - Planning time spent working on the project and organising when I will be working on each section of the project is paramount to its success. The plans success will be decided by how far I enforce it across the projects lifespan. By planning what I need to work on when, I will have a better idea of how much I have accomplished after each deadline. If the time limit of the project starts to become an issue, I will have to review the requirements of the final application to see if there is anything that can be missed that won't be too detrimental to the success of the project.

First Indicator - The first indicator that the time limit of the project is likely to impact the progress of the project is when I will have missed one of the deadlines set out in my project plan. If this occurs, I will need to re-address my plan.

8.2 Illness

Description - Illness can affect the time management of the project as I may be unable to work on the project while sick.

Risk Impact - If at any time during the project I am unwell enough to be unable to work on the project, it may affect the final deliverables and whether the project can be submitted in time for the final deadline.

Mitigation / Control - I must maintain a healthy lifestyle whilst working on the project to minimise the risk of myself getting unwell. In the unlikely event that I am unwell enough to be unable to work on the project for a significant period of time, I will make sure to contact my supervisor for advice on how to proceed. If an illness hinders me from working for a short period of time, I will have to readdress my plan.

First Indicator - The first indicator of this becoming a risk to the project will be my declining health.

8.3 Loss of Work

Description - Loss of work has the potential to have a significant impact on the final deliverables of the project.

Risk Impact - Loss of work has the potential of undoing a significant portion of the work I have put into the project.

Mitigation / Control - I am mitigating this risk by keeping online backups of all my work on GitHub and Overleaf.

First Indicator -

9 Starting point of research

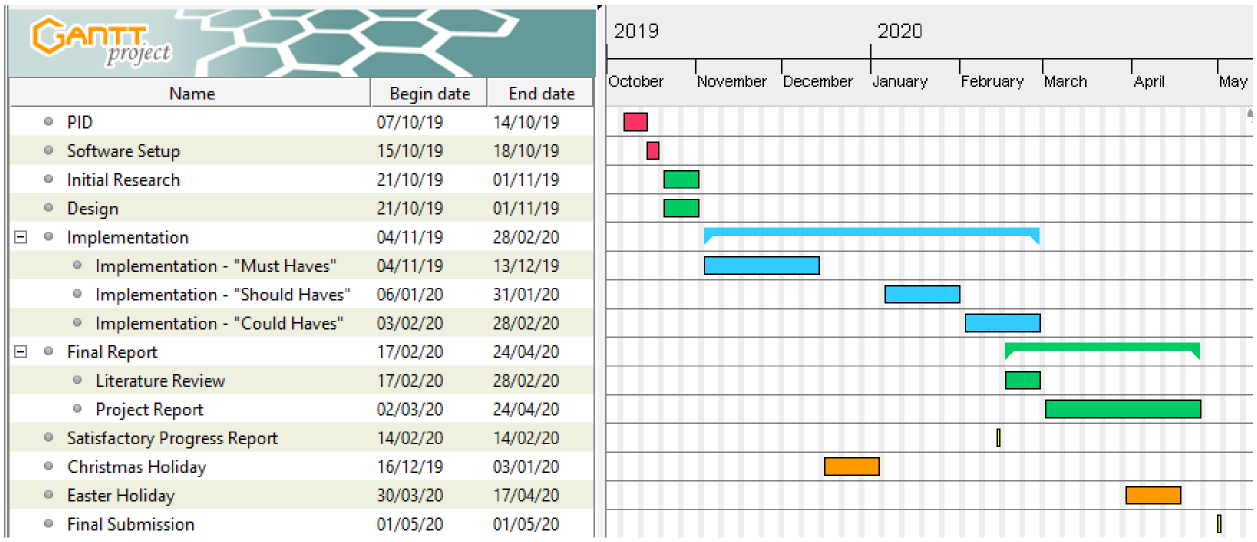
To start my research to develop the most successful solution I can, I will firstly research Parsons Problems and its positive and negative impacts on teaching programming. By reviewing their negative impact, I can make sure that my project does not share the same negative connotation.

I will also explore other interactive tools online. This will include sites such as `www.codewars.com`. Though CodeWars approach to teaching or challenging users on their programming skill is different to that of Parsons Problems, I will investigate its method of presenting problems to the users and suiting its given problems to an individual.

I will also test out other programming teaching apps available on the Google Play Store for Android to observe other potential approaches to teaching novice programmers. I can also observe how they structure the problems they give to their users, their software design and layout to determine whether I can improve on them in my project.

10 Project plan

Appended to this document will be a copy of the project plan I have put together. I have decided to break down the implementation period of my project into three sections, each one correlating to how I have categorised my requirements (”Must Dos”, ”Should Dos” and ”Could Dos”).



Name	Begin date	End date
PID	07/10/19	14/10/19
Software Setup	15/10/19	18/10/19
Initial Research	21/10/19	01/11/19
Design	21/10/19	01/11/19
Implementation	04/11/19	28/02/20
Implementation - "Must Haves"	04/11/19	13/12/19
Implementation - "Should Haves"	06/01/20	31/01/20
Implementation - "Could Haves"	03/02/20	28/02/20
Final Report	17/02/20	24/04/20
Literature Review	17/02/20	28/02/20
Project Report	02/03/20	24/04/20
Satisfactory Progress Report	14/02/20	14/02/20
Christmas Holiday	16/12/19	03/01/20
Easter Holiday	30/03/20	17/04/20
Final Submission	01/05/20	01/05/20

Figure 1: Project Plan

11 Legal, ethical, professional and social issues

I currently have no plans to complete any user testing during my project and should therefore not require any collection of personal data. I do not require any ethical approval to complete my project. There are no constraints on my project imposed by any professional, social or legal issues.

Certificate of Ethics Review

Project Title: Mobile App for Learning Basic Algorithm Construction

Name: Jack David Turner **User ID:** 776193 **Application Date:** 11-Oct-2019 13:33 **ER Number:** ETHIC-2019-1033

You must download your referral certificate, print a copy and keep it as a record of this review.

The FEC representative for the School of Computing is Carl Adams

It is your responsibility to follow the University Code of Practice on Ethical Standards and any Department/School or professional guidelines in the conduct of your study including relevant guidelines regarding health and safety of researchers including the following:

- University Policy
- Safety on Geological Fieldwork

It is also your responsibility to follow University guidance on Data Protection Policy:

- General guidance for all data protection issues
- University Data Protection Policy

Which school/department do you belong to?: **SOC**
What is your primary role at the University?: **UndergraduateStudent**
What is the name of the member of staff who is responsible for supervising your project?: **Matthew Poole**
Is the study likely to involve human subjects (observation) or participants?: **No**
Are there risks of significant damage to physical and/or ecological environmental features?: **No**
Are there risks of significant damage to features of historical or cultural heritage (e.g. impacts of study techniques, taking of samples)?: **No**
Does the project involve animals in any way?: **No**
Could the research outputs potentially be harmful to third parties?: **No**
Could your research/artefact be adapted and be misused?: **No**
Does your project or project deliverable have any security implications?: **No**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**
Please read and confirm that you agree with the following statements: **Confirmed**

Supervisor Review

As supervisor, I will ensure that this work will be conducted in an ethical manner in line with the University Ethics Policy.

Supervisor signature: [illegible] **Date:** 11/10/19